Problem Decomposition for Traffic Diversions

Quanshi Xia[1], Andrew Eremin[1], and Mark Wallace[2]

[1] IC-Parc, Imperial College London, London SW7 2AZ, UK
{q.xia, a.eremin}@imperial.ac.uk
[2] School of Business Systems, Monash University, Clayton, Vic 3800, Australia
mark.wallace@infotech.monash.edu.au

Abstract. When a major road traffic intersection is blocked, vehicles should be diverted from the incoming roads in such a way as to avoid the roads on the diversions from also becoming over-congested. Assuming different diversions may use partly the same roads, the challenge is to satisfy the following traffic flow constraint: ensure that even in the worst case scenario, the diversions can accommodate the same volume of traffic as the blocked intersection.
The number of diversions increases quadratically with the number of roads at the intersection. Moreover any road may be used by any subset of the diversions - thus the number of *worst cases* can grow exponentially with the number of diversions.
This paper investigates two different approaches to the problem, describes their implementation on the hybrid MIP/CP software platform ECL^{i}PS^e, and presents benchmark results on a set of test cases.

1 Introduction

Cities are becoming more congested, but luckily road management technology - sensing, signs, lights *etc.* - is improving dramatically. We now have the opportunity to apply planning and optimisation techniques to road management to reduce congestion and optimise journey times.

The problem of diversions tackled in this paper is an artificial one, in that some of the assumptions do not hold on the ground. However the problem appears in the context of a larger system for traffic management, and its solution is in practical use today.

The problem focuses on planning diversions to get around a blocked junction or interchange, where a number of routes meet each other. Assuming no information about the destinations of vehicles on the road, the aim is to ensure that every incoming route is linked to every outgoing route by a diversion.

However the requirement is also to ensure that whatever traffic might have been flowing through the junction, the diversion routes are sufficiently major to cope with them. For the purposes of this problem, we ignore any traffic that happens to be using the diversion roads for other journeys, that would not have passed through the blocked junction.

The problem is scientifically interesting because, until all the diversions have been specified, it is not possible to tell what is the maximum possible traffic flow that could be generated along any given road on a diversion.

Let us illustrate this with an example: The junction **j** has three incoming

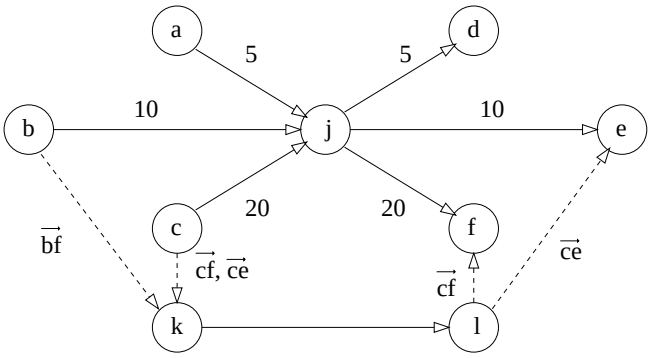

Fig. 1. A Simple Junction

roads, from **a**, **b** and **c** and three outgoing, to **d**, **e** and **f**. Each road has a grade, which determines the amount of traffic it can carry. These amounts are 5 for $\boldsymbol{aj}$ and $\boldsymbol{jd}$, 10 for $\boldsymbol{bj}$ and $\boldsymbol{je}$ and 20 for $\boldsymbol{cj}$ and $\boldsymbol{jf}$.

The diversion $\boldsymbol{cf}$ from **c** to **f** clearly needs to be able to carry a traffic quantity of 20. Assume that this diversion shares a road $\boldsymbol{kl}$ with the diversion $\boldsymbol{bf}$ from **b** to **f**. The total traffic on both diversions, in the worst case, is still only 20 because the diverted routes both used the road $\boldsymbol{jf}$, which has a capacity of 20.

However if the diversion $\boldsymbol{ce}$ from **c** to **e** also uses the road $\boldsymbol{kl}$, then in the worst case the traffic over $\boldsymbol{kl}$ goes up to 30. This case arises when there is a flow of 10 from **b** to **j**, a flow of 20 from **c** to **j**, a flow of 10 from **j** to **e**, and another flow of 20 from **j** to **f**. This means that there may potentially be end-to-end flows of 10 from **b** to **f**, from **c** to **e** and from **c** to **f**.

The total number of diversions that must be set up in this case is 9, a diversion from each incoming origin to each outgoing destination. In general, then, the number of diversions grows quadratically in the size of the junction. Moreover any subset of these diversions may intersect, so the number of worst case scenarios to be calculated is potentially exponential in the number of diversions!

The final aspect of the problem is to find actual routes for all the diversions, which satisfy all the worst case scenarios. Given all the above possibilities, a generate and test approach may need to explore a huge number of routes.

In this paper we present several approaches to solving the problem. The first is a global integer/linear model, which can solve smaller problem instances but grows in memory usage and execution time for larger problem instances which limits its scalability. The next three are increasingly sophisticated versions of a problem decomposition. The efficient handling of the master and subproblems, and addition of new rows to the master problem are supported by the ECL^{i}PS^e constraint programming system. The most sophisticated model solves all the benchmark problem instances with less than 50 iterations.

2 Problem Model

The road network is broken down into junctions, and road segments connecting them. Each diversion is mapped to a path from the origin to the destination, avoiding the blocked junction. To model the block, we simply drop the junction and its connected roads from the network.

The challenge is to model the capacity constraints on each road segment in each path in the network. For each road segment, the sum of the traffic flows on all the routes whose diversions pass through that road segment must be accommodated. The road segment is over-congested if, in the worst case, this sum exceeds the capacity of the road segment.

In this paper we shall write constants in lower case (e.g. e, *edge*), we shall write variables starting with an upper case letter (e.g. Q, *Quantity*), variable arrays are subscripted (e.g. with a single subscript, Q_e, or with multiple subscripts Q_{fe}), we shall write functions using brackets (e.g. `dest`*(f)*). We use **bold** identifiers to denote sets (e.g. $\mathbf{E}$). For example, to say that edge e belongs to the set of edges $\mathbf{E}$, we write $e \in \mathbf{E}$. Set-valued variables and functions are also written in **bold** font.

We formalism the problem in terms of a network, with *edges* representing road segments, and *nodes* representing junctions.

The network comprises a set of edges, $\mathbf{E}$ and a set of nodes, $\mathbf{N}$. Each edge $e \in \mathbf{E}$ has a capacity `cap`*(e)*. Allowing for one-way traffic, we associate a direction with each edge (two way roads are therefore represented by two edges, one in each direction). The edge from origin o into the junction has capacity `ocap`*(o)* and the edge leaving the junction and entering the destination d has capacity `dcap`*(d)*.

For each node n there is a set of edges $\mathbf{IN}(n)$ entering n and a set of edges $\mathbf{OUT}(n)$ leaving n.

The set of traffic flows to be diverted is $\mathbf{F}$. Each flow $f \in \mathbf{F}$ has an origin `orig`*(f)*, a destination `dest`*(f)* and a maximum flow quantity `quan`*(f)*.

`quan`*(f)* is limited by the size of the roads of the diverted flow, into the junction from the origin and out from the junction to the destination. Thus, $\texttt{quan}(f) = min\{\texttt{ocap}(\texttt{orig}(f)), \texttt{dcap}(\texttt{dest}(f))\}$.

The diversion for the flow f is a path $\mathbf{DIV}_f$ joining its origin `orig`*(f)* to its destination `dest`*(f)*. (Assuming no cycles, we model the path as a set of edges, thus $\mathbf{DIV}_f$ is a set-valued variable.)

The awkward constraints are the capacity constraints. For this purpose we elicit the worst case for each edge, using an optimisation function.

Consider the total flow diverted through an edge e: for each flow f there is a non-negative flow quantity $Q_{fe} \leq \texttt{quan}(f)$ diverted through e. For all flows f that are not diverted through the edge e this quantity is 0, while for all flows in the set of flows diverted through an edge e, $\mathbf{F}_e = \{f : e \in \mathbf{DIV}_f\}$, there is a non-negative flow quantity.

The total diverted flow DQ_e through the edge e is therefore $DQ_e = \sum_{f \in \mathbf{F}_e} Q_{fe}$. Clearly it must be within the edge capacity: $\texttt{cap}(e) \geq DQ_e$.

The maximum total diverted flow through an edge is in general less than the sum of the maxima, $\texttt{quan}(f)$ of all the individual flows. Indeed the maximum quantity of the *sum* of *all* the flows which have the same origin o is constrained by $\texttt{ocap}(o) \geq \sum_{f:\texttt{orig}(f)=o} Q_{fe}$. Similarly for destination d: $\texttt{dcap}(d) \geq \sum_{f:\texttt{dest}(f)=d} Q_{fe}$.

The worst case for capacity constraint on edge e is when DQ_e is maximized, by changing the flows through the original junction. The resulting constraint is $\texttt{cap}(e) \geq \max_{Q_{fe}} \sum_{f \in \mathbf{F}_e} Q_{fe}$.

3 Formulation as a MIP Problem

For the MIP model binary (0/1) variables X_{fe} and continuous variables Q_{fe} are introduced. For each flow f and edge e, $X_{fe} = 1$ if and only if flow f is diverted through edge e. Thus, $\mathbf{DIV}_f = \{e : X_{fe} = 1\}$

The problem is to choose diversions (by setting the values of the variables X_{fe}) such that all the worst case capacity constraints are satisfied. We introduce an optimisation expression: $\min_{X_{fe}} \sum_{f \in \mathbf{F}} \sum_{e \in \mathbf{E}} X_{fe}$, which precludes cycles in any diversion since optimisation would set the flow through any cycle to zero and minimizes the total diversion path length.

$$
\begin{array}{l}
\min\limits_{X_{fe}} \quad \sum\limits_{f \in \mathbf{F}} \sum\limits_{e \in \mathbf{E}} X_{fe} \\
st. \quad \left\{ \begin{array}{ll}
\forall f \in \mathbf{F}: & \left\{ \begin{array}{l}
\forall n \in \mathbf{N} \setminus \{\texttt{orig}(f), \texttt{dest}(f)\} : \sum\limits_{e \in \mathbf{IN}(n)} X_{fe} = \sum\limits_{e \in \mathbf{OUT}(n)} X_{fe} \\
n = \texttt{orig}(f) : \sum\limits_{e \in \mathbf{OUT}(n)} X_{fe} = 1 \\
n = \texttt{dest}(f) : \sum\limits_{e \in \mathbf{IN}(n)} X_{fe} = 1
\end{array} \right. \\
\forall e \in \mathbf{E}: & \texttt{cap}(e) \geq \left\{ \begin{array}{ll}
\max\limits_{Q_{fe}} & \sum\limits_{f \in \mathbf{F}} X_{fe} * Q_{fe} \\
st. & \left\{ \begin{array}{l}
\forall o \in \texttt{orig}(\mathbf{F}) : \texttt{ocap}(o) \geq \sum\limits_{f:\texttt{orig}(f)=o} Q_{fe} \\
\forall d \in \texttt{dest}(\mathbf{F}) : \texttt{dcap}(d) \geq \sum\limits_{f:\texttt{dest}(f)=d} Q_{fe}
\end{array} \right.
\end{array} \right. \\
X_{fe} \in \{0, 1\}, \quad \texttt{quan}(f) \geq Q_{fe} \geq 0
\end{array} \right.
\end{array}
\tag{1}
$$

The embedded optimisation for each edge e can be linearized by using the Karush-Kuhn-Tucker condition [1]. First we dualise it, introducing dual variables D_{oe} and D_{de}

$$
\texttt{cap}(e) \geq \left\{ \begin{array}{ll}
\min\limits_{D_{oe}, D_{de}} & \sum\limits_{o \in \texttt{orig}(\mathbf{F})} \texttt{ocap}(o) * D_{oe} + \sum\limits_{d \in \texttt{dest}(\mathbf{F})} \texttt{dcap}(d) * D_{de} \\
st. & \left\{ \begin{array}{ll}
\forall f \in \mathbf{F}, o = \texttt{orig}(f), d = \texttt{dest}(f) : & D_{oe} + D_{de} \geq X_{fe} \\
D_{oe} \in \{0, 1\}, \quad D_{de} \in \{0, 1\}
\end{array} \right.
\end{array} \right.
\tag{2}
$$

Note that the upper bounds on the variables Q_{fe} are implicit from the origin and destination constraints and variable non-negativity; in forming the dual problem

we have dropped these redundant bounds. Further since the coefficients of the variables D_{oe}, D_{de} in the cost function to be minimized in the dual are strictly positive and the variables non-negative an upper bound of 1 can be deduced for the value of all dual variables in any dual optimal solution, and thus in any feasible solution to the original problem, from the dual constraints and the upper bounds of X_{fe}. Moreover since the constraints of (2) are totally unimodular any basic feasible solution and hence any basic optimal feasible solution is integral, and D_{oe}, D_{de} reduce to binary variables.

We introduce slack variables SQ_{oe} and SQ_{de} for the constraints in the primal, and dual slack variables SD_{fe} in the dual. We can now replace the embedded maximization problem for each edge e by the following constraints:

$$\begin{cases} \mathtt{cap}(e) \geq \sum_{f \in \mathbf{F}} Q_{fe} \\ \forall o \in \mathtt{orig}(\mathbf{F}): \quad \mathtt{ocap}(o) = \sum_{f:\mathtt{orig}(f)=o} Q_{fe} + SQ_{oe} \\ \forall d \in \mathtt{dest}(\mathbf{F}): \quad \mathtt{dcap}(d) = \sum_{f:\mathtt{dest}(f)=d} Q_{fe} + SQ_{de} \\ \forall f \in \mathbf{F}, o = \mathtt{orig}(f), d = \mathtt{dest}(f): \quad D_{oe} + D_{de} - SD_{fe} = X_{fe} \\ \left.\begin{array}{l} \forall f \in \mathbf{F}: \quad Q_{fe} * SD_{fe} = 0 \\ \forall o \in \mathtt{orig}(\mathbf{F}): \quad SQ_{oe} * D_{oe} = 0 \\ \forall d \in \mathtt{dest}(\mathbf{F}): \quad SQ_{de} * D_{de} = 0 \end{array}\right\} \text{ complementarity} \\ X_{fe} \in \{0,1\}, \quad D_{oe} \in \{0,1\}, \quad D_{de} \in \{0,1\}, \quad SD_{fe} \in \{0,1\} \\ \mathtt{quan}(f) \geq Q_{fe} \geq 0, \quad \mathtt{ocap}(o) \geq SQ_{oe} \geq 0, \quad \mathtt{dcap}(d) \geq SQ_{de} \geq 0 \end{cases} \tag{3}$$

Since D_{oe}, D_{de}, SD_{fe} are binary (0/1) variables, the complementarity constraints can be linearized to obtain the mixed integer linear programming model which can be solved by MIP solvers:

$$\min_{X_{fe}} \sum_{f \in \mathbf{F}} \sum_{e \in \mathbf{E}} X_{fe}$$

$$st. \quad \begin{cases} \forall f \in \mathbf{F}: \begin{cases} \forall n \in \mathbf{N} \setminus \{\mathtt{orig}(f), \mathtt{dest}(f)\}: \sum_{e \in \mathbf{IN}(n)} X_{fe} = \sum_{e \in \mathbf{OUT}(n)} X_{fe} \\ n = \mathtt{orig}(f): \sum_{e \in \mathbf{OUT}(n)} X_{fe} = 1 \\ n = \mathtt{dest}(f): \sum_{e \in \mathbf{IN}(n)} X_{fe} = 1 \end{cases} \\ \forall e \in \mathbf{E}: \begin{cases} \mathtt{cap}(e) \geq \sum_{f \in \mathbf{F}} Q_{fe} \\ \forall o \in \mathtt{orig}(\mathbf{F}): \quad \mathtt{ocap}(o) = \sum_{f:\mathtt{orig}(f)=o} Q_{fe} + SQ_{oe} \\ \forall d \in \mathtt{dest}(\mathbf{F}): \quad \mathtt{dcap}(d) = \sum_{f:\mathtt{dest}(f)=d} Q_{fe} + SQ_{de} \\ \forall f \in \mathbf{F}, o = \mathtt{orig}(f), d = \mathtt{dest}(f): D_{oe} + D_{de} - SD_{fe} = X_{fe} \\ \forall f \in \mathbf{F}: \quad \mathtt{quan}(f)(1 - SD_{fe}) - Q_{fe} \geq 0 \\ \forall o \in \mathtt{orig}(\mathbf{F}): \quad \mathtt{ocap}(o)(1 - D_{oe}) - SQ_{oe} \geq 0 \\ \forall d \in \mathtt{dest}(\mathbf{F}): \quad \mathtt{dcap}(d)(1 - D_{de}) - SQ_{de} \geq 0 \end{cases} \\ X_{fe} \in \{0,1\}, \quad D_{oe} \in \{0,1\}, \quad D_{de} \in \{0,1\}, \quad SD_{fe} \in \{0,1\} \\ \mathtt{quan}(f) \geq Q_{fe} \geq 0, \quad \mathtt{ocap}(o) \geq SQ_{oe} \geq 0, \quad \mathtt{dcap}(d) \geq SQ_{de} \geq 0 \end{cases} \tag{4}$$

The resulting performance is summarized in Table 1 under the column of **MIP**.

4 Formalization Using Decomposition

Most real resource optimisation problems involve different kinds of constraints, which are best handled by different kinds of algorithms. Our traffic diversions problem can be decomposed into parts which are best handled by different constraint solvers.

Both different problem decompositions and the use of different solvers for the resulting components were tried on this problem. For reasons of space, we present just one decomposition, into a master problem and a set of (similar) subproblems. The master problem is a pure integer linear programming which is best handled by a MIP solver. The subproblems are very simple linear programs and well-suited to a linear solver, although they could equally be solved by CP as in [2, 3]. In our approach, CP provides the modelling language and the glue which enables the solvers to communicate, though not the solvers themselves.

4.1 Informal Description of the Decomposition

The original problem (1) can be treated instead by decomposing it into a multi-commodity flow master problem, and a maximization subproblem for each edge in the network.

The master problem simply assigns a path to each flow. Initially these paths are independent. However as a result of solving the subproblems new constraints are, later, added to the master problem which preclude certain combinations of flows from being routed through the same edge.

Each subproblem takes as an input the path assigned to each flow by the latest solution of the master problem. If the edge associated with the subproblem is e, the relevant flows $\mathbf{F}_e$ are those whose paths are routed through edge e. The subproblem then maximizes the sum of the flows in $\mathbf{F}_e$. If this maximum sum exceeds the capacity of the edge, then a new constraint is created and passed back to the master problem precluding any assignment which routes all the flows in $\mathbf{F}_e$ through the edge e. Although the cuts added to the master problem are formed differently the principle behind this approach is closely related to that of classic Benders decomposition [4] or its logic-based extension [5].

4.2 Model Specification

The formalization of this decomposed model uses the same binary variables X_{fe} as the MIP model. Each time the master problem is solved it assigns values (0 or 1) to all these binary variables. For the assignment to X_{fe} returned by the solution of the k^{th} master problem, we write x^k_{fe}.

The subproblems in the current model are linear maximization problems of the kind that typically occurs in production planning, which use the same continuous variables Q_{fe} as the original problem formulation in Section 2 above.

Accordingly the k^{th} subproblem associated with edge e is simply:

$$\max_{Q_{fe}} \sum_{f \in \mathbf{F}} x_{fe}^k * Q_{fe}$$

$$st. \begin{cases} \forall o \in \mathtt{orig}(\mathbf{F}): \quad \mathtt{ocap}(o) \geq \sum_{f:\mathtt{orig}(f)=o} Q_{fe} \\ \forall d \in \mathtt{dest}(\mathbf{F}): \quad \mathtt{dcap}(d) \geq \sum_{f:\mathtt{dest}(f)=d} Q_{fe} \\ \mathtt{quan}(f) \geq Q_{fe} \geq 0 \end{cases} \tag{5}$$

The solution to the k^{th} subproblem associated with edge e, is a set of flow quantities, which we can write as q_{fe}^k for each flow f.

Suppose the subproblem associated with edge e indeed returns a maximum sum of flows which exceeds $\mathtt{cap}(e)$, *i.e.* $\sum_{f \in \mathbf{F}} q_{fe}^k > \mathtt{cap}(e)$. Then the constraint passed to the $(k+1)^{th}$ master problem from this subproblem is

$$\sum_{f \in \mathbf{F}} x_{fe}^k * (1 - X_{fe}) \geq 1 \tag{6}$$

This constraint ensures that at least one of the flows previously routed through edge e will no longer be routed through e. Therefore, it simply rules out the previous assignment and those assignments with previous assignment as the subset [2].

The k^{th} master problem has the form:

$$\min_{X_{fe}} \sum_{f \in \mathbf{F}} \sum_{e \in \mathbf{E}} X_{fe}$$

$$st. \begin{cases} \forall f \in \mathbf{F}: \begin{cases} \forall n \in \mathbf{N} \setminus \{\mathtt{orig}(f), \mathtt{dest}(f)\}: \sum_{e \in \mathbf{IN}(n)} X_{fe} = \sum_{e \in \mathbf{OUT}(n)} X_{fe} \\ n = \mathtt{orig}(f): \sum_{e \in \mathbf{OUT}(n)} X_{fe} = 1 \\ n = \mathtt{dest}(f): \sum_{e \in \mathbf{IN}(n)} X_{fe} = 1 \end{cases} \\ \text{for certain edges } e \text{ and iterations } j < k: \sum_{f \in \mathbf{F}} x_{fe}^j * (1 - X_{fe}) \geq 1 \\ X_{fe} \in \{0, 1\} \end{cases} \tag{7}$$

This model can be solved by completing a branch and bound search at every iteration of the master problem, in order to return the shortest feasible paths, satisfying all the cuts returned from earlier subproblems. If only feasible, rather than shortest, diversions are required, optimality of the master problem solution is not necessary, and the master problem solution can be stopped as soon as an integer feasible solution is found. However, the path constraints, as they stand, admit non-optimal solutions in which there might be cyclic sub-paths in (or even disjoint from) the path. Whilst it is not incorrect to admit such irrelevant cyclic sub-paths, in fact such cycles can easily be eliminated by a pre-processing step between master and subproblem solution since the path produced by removing

cycles from a feasible solution to the k^{th} master problem remains feasible. Such a pre-processing step would make the diversion problem be solved more efficiently.

After the current master problem returns a feasible solution, it is then checked by running one or more subproblems, associated with different edges. Naturally if none of the subproblems produced a maximum flow which exceeded the capacity of its edge, then the master problem solution is indeed a solution to the original diversion problem. In this case the algorithm succeeds. If, on the other hand, after a certain number of iterations, the master problem has no feasible solution then the original diversion problem is unsatisfiable. There is no way of assigning diversions to flows that have the capacity to cope with the worst case situation.

The experimental evaluation of this algorithm is given in Table 1 under the column of **D(naive)**.

5 An Enhanced Decomposition

Under certain circumstances the previous decomposition leads to a very large number of iterations of the master problem, with many cuts added during the iterations. The result is that the master problem becomes bigger and more difficult to solve. Moreover, the master problem has to be solved by branch and bound search at each of a large number of iterations. This has a major impact on run times, as shown in column **D(naive)** of the experiments in Table 1.

5.1 Generating Fewer Cuts

A cut that only removes the previous assignment is easy to add, but typically not very strong: a different cut has to be added for each assignment that is ruled out. This may require a very large number of cuts. Instead, for the diversion problem, one could reduce the number of cuts by considering the flow quantities of the diverted flows whose diversion is routed by the k^{th} master problem solution through the relevant edge.

Now instead of posting the cut (6) which simply rules out the previous assignment of diversions to edge e, we can explicitly use the flow quantities and return the constraint

$$\sum_{f \in \mathbf{F}} q^k_{fe} * X_{fe} \leq \mathtt{cap}(e) \tag{8}$$

Using this set of cuts, the k^{th} master problem then has the form:

$$\min_{X_{fe}} \sum_{f \in \mathbf{F}} \sum_{e \in \mathbf{E}} X_{fe}$$

$$st. \begin{cases} \forall f \in \mathbf{F}: & \begin{cases} \forall n \in \mathbf{N} \setminus \{\mathtt{orig}(f), \mathtt{dest}(f)\}: \sum_{e \in \mathbf{IN}(n)} X_{fe} = \sum_{e \in \mathbf{OUT}(n)} X_{fe} \\ n = \mathtt{orig}(f): \sum_{e \in \mathbf{OUT}(n)} X_{fe} = 1 \\ n = \mathtt{dest}(f): \sum_{e \in \mathbf{IN}(n)} X_{fe} = 1 \end{cases} \\ \text{for certain edges } e \text{ and iterations } j < k: \sum_{f \in \mathbf{F}} q^j_{fe} * X_{fe} \leq \mathtt{cap}(e) \\ X_{fe} \in \{0, 1\} \end{cases} \tag{9}$$

The resulting performance is summarized in Table 1 under the column of $\mathbf{D(0)}$.

5.2 Generating Tighter Cuts

The optimisation function in (5) gives zero weight to any flows $f \notin \mathbf{F}_e$, for which $x^k_{fe} = 0$. For any optimal subproblem solution with $q^k_{fe} > 0$ for some $f \notin \mathbf{F}_e$ there exists an equivalent optimal solution with $q^k_{fe} = 0$. Thus the flow quantities q^k_{fe} $(\forall f \notin \mathbf{F}_e)$ in optimal solutions to the subproblem may be zero rather than non-zero. The cut (8) thus may only constrain variables X_{fe} for which $x^k_{fe} = 1$.

Instead, for the diversion problem, one could reduce the number of cuts by considering the flow quantities of *all* the diverted flows, not just the ones whose diversion is routed by the k^{th} master problem solution through the relevant edge.

To extract the tightest cut from this subproblem, we therefore change the optimisation function so as to first optimise the flow through the relevant edge, and then, for any other flows which are still free to be non-zero, to maximize those flows too. This is achieved by simply adding a small multiplier ϵ to the other flows in the optimisation function:

$$\max_{Q_{fe}} \sum_{f \in \mathbf{F}} (x^k_{fe} + \epsilon) * Q_{fe}$$

$$st. \begin{cases} \forall o \in \mathtt{orig}(\mathbf{F}): \quad \mathtt{ocap}(o) \geq \displaystyle\sum_{f:\mathtt{orig}(f)=o} Q_{fe} \\ \forall d \in \mathtt{dest}(\mathbf{F}): \quad \mathtt{dcap}(d) \geq \displaystyle\sum_{f:\mathtt{dest}(f)=d} Q_{fe} \\ \mathtt{quan}(f) \geq Q_{fe} \geq 0 \end{cases} \tag{10}$$

Now all the variables Q_{fe} will take their maximum possible values (denoted as $\tilde{q}^k_{fe}$), ensuring that (8) expresses as tight a cut as possible.

$$\sum_{f \in \mathbf{F}} \tilde{q}^k_{fe} * X_{fe} \leq \mathtt{cap}(e) \tag{11}$$

This cut not only constrains variables X_{fe} for which $x^k_{fe} = 1$, but also constrains the value of X_{fe} for other flows f which may not have used this edge in the k^{th} subproblem.

Accordingly the k^{th} master problem then has the form:

$$\min_{X_{fe}} \sum_{f \in \mathbf{F}} \sum_{e \in \mathbf{E}} X_{fe}$$

$$st. \begin{cases} \forall f \in \mathbf{F}: \begin{cases} \forall n \in \mathbf{N} \setminus \{\mathtt{orig}(f), \mathtt{dest}(f)\}: \displaystyle\sum_{e \in \mathbf{IN}(n)} X_{fe} = \sum_{e \in \mathbf{OUT}(n)} X_{fe} \\ n = \mathtt{orig}(f): \displaystyle\sum_{e \in \mathbf{OUT}(n)} X_{fe} = 1 \\ n = \mathtt{dest}(f): \displaystyle\sum_{e \in \mathbf{IN}(n)} X_{fe} = 1 \end{cases} \\ \text{for certain edges } e \text{ and iterations } j < k: \displaystyle\sum_{f \in \mathbf{F}} \tilde{q}^j_{fe} * X_{fe} \leq \mathtt{cap}(e) \\ X_{fe} \in \{0, 1\} \end{cases} \tag{12}$$

The experimental results on the enhanced decomposition model are given in Table 1 under the column of $\mathbf{D}(\epsilon)$, where $\epsilon = 1.0e^{-5}$.

5.3 Comparison of Cuts Tightness

The 3 different cut generation formulations, (6),(8) and (11) have been presented. The tightness for these cuts, generated by different cut formulations are different too. For simple example, supposed that the assignment to X_{fe} returned by the solution of the k^{th} master problem as

$$[x^k_{1e}, x^k_{2e}, x^k_{3e}, x^k_{4e}, x^k_{5e}] = [1, 1, 1, 0, 0]$$

The flow quantities Q_{fe} returned by the k^{th} subproblem (5) solution as

$$[q^k_{1e}, q^k_{2e}, q^k_{3e}, q^k_{4e}, q^k_{5e}] = [50, 75, 75, 50, 0]$$

and one returned by the solution of the k^{th} subproblem (10) as

$$[\tilde{q}^k_{1e}, \tilde{q}^k_{2e}, \tilde{q}^k_{3e}, \tilde{q}^k_{4e}, \tilde{q}^k_{5e}] = [50, 75, 75, 50, 75]$$

Notice that q^k_{4e} can also take 0 as optimal subproblem (5) solution because $x^k_{4e} = 0$.

By using of the cut generation formulation (6) we will obtain a cut

$$X_{1e} + X_{2e} + X_{3e} \leq 2$$

and the cut generated by the cut formulation (8) is

$$50 * X_{1e} + 75 * X_{2e} + 75 * X_{3e} + 50 * X_{4e} \leq 100$$

and the cut formulation (11) generated a cut of

$$50 * X_{1e} + 75 * X_{2e} + 75 * X_{3e} + 50 * X_{4e} + 75 * X_{5e} \leq 100$$

It is trivial to show that these cuts are getting tighter and tighter!

6 Implementation

The problem was solved using the ECL^{i}PS^e constraint programming platform [6]. ECL^{i}PS^e provides interfaces both to CPLEX [7] and to Xpress-MP [8] for solving MIP problems. The current application comprises either a single MIP problem for the model presented in Section 3 or for the decomposed models presented in Sections 4–5 an MIP master problem and, in principle, LP subproblems for each edge in the network.

The MIP master problem and LP subproblems were run as separate external solver *instances* using the ECL^{i}PS^e facility to set up multiple external solver matrices. ECL^{i}PS^e enables different instances to be modified and solved at will. However the implementation does not run - or even set up - subproblems associated with edges which do not lie on any path returned by the master problem since the capacity constraints on these edges cannot, of course, be exceeded.

In the implementation we may choose how many cuts to return to the master problem after each iteration. One extreme is to run the subproblems individually and stop as soon as a new cut has been elicited while the other is to run all subproblems. These choices will respectively result in only one cut or all possible cuts being added to the master problem at each iteration. The additional cuts will tend to result in fewer iterations at the cost of more time spent per iteration on subproblem solution.

Preliminary experiments showed that it was substantially more efficient in general to run all the subproblems and add all the cuts at every iteration. This result is unsurprising for the current application since the cost of solving the master problem far outweighs that for each subproblem. The subproblems for each edge are LPs involving $|\mathbf{F}|$ variables and $|\texttt{orig}(\mathbf{F})| + |\texttt{dest}(\mathbf{F})|$ constraints, and are solved very quickly. The master problem however is a pure integer problem involving $|\mathbf{F}| \times |\mathbf{E}|$ variables and $|\mathbf{F}| \times |\mathbf{N}|$ constraints plus any cuts added so far. The initial master problem constraints are totally unimodular. As cuts are added in further iterations the unimodularity of the master problem is destroyed requiring solution by branch-and-bound search.

The ECL^{i}PS^e interface enables the user to solve a single problem instance using different optimisation functions. In the current application, the subproblem associated with each edge and each iteration has the same constraints. The difference lies only in the optimisation function. Therefore, only one subproblem needs to be set up when the algorithm is initialized: our implementation uses the same problem instance for each subproblem, simply running it with a different optimisation function for each edge.

Accordingly, in our ECL^{i}PS^e implementation, the decomposition model was set up using two different problem instances, one for the master problem and one for all the subproblems. In the following ECL^{i}PS^e code, these instances are named `master` and `sub`. First we create the master problem and the subproblem template:

```
:- eplex_instance(master).                                  %1
:- eplex_instance(sub).                                     %2

diversions(Xfe_Array) :-                                    %3
    % Multicommodity flow constraints written here         %4
    master:eplex_solver_setup(min(sum(Xfe_Array))),         %5

    % Subproblem constraints written here                   %6
    sub:eplex_solver_setup(max(sum(Qfe_Array))),            %7

    iterate(Xfe_Array, Qfe_Array).                          %8
```

At lines 1 and 2 a master and subproblem solver instance are declared. They will later be filled with linear (and integer) constraints. Line 3 names the procedure that the user can invoke to solve the problem.

The problem constraints are entered next. These constraints are problem-instance-specific. Typically they are automatically generated from a data file holding the details of the network.

Line 5 sets up the master problem, reading in the constraints added previously, and posting a default optimisation function (minimize the total number of edges used in all the diversions).

The problem-instance-specific edge capacity constraints are now entered. Then the subproblem is set up at line 7, again with a default optimisation function. Finally at line 8 the iteration procedure is invoked.

We now write down the code which controls the iteration.

```
iterate(Xfe_Array, Qfe_Array) :-                                    %9
    master:eplex_solve(Cost),                                       %10
    master:eplex_get(typed_solution,Xfe_Values),                    %11
    % Find 'Edges' used in paths                                    %12
    ( foreach(E,Edges),                                             %13
      % Collect violated capacity constraints into 'CutList'       %14
    do                                                              %15
        % Build optimisation expression 'Expr'                      %16
        sub:eplex_probe(max(Expr),SubCost),                         %17
        ( SubCost > cap(E)  ->  % Create cut                        %18
        )
    ),
    ( foreach(Cut, CutList)                                         %19
    do                                                              %20
        master:Cut  % Add cut                                       %21
    ),
    % If CutList is empty, succeed, otherwise iterate again        %22
```

At each iteration, the program solves the master problem (line 10), extracts the solution (values of all the X_{fe} variables), (line 11), finds which edges are on diversions (code not given) and then builds, for each edge (line 13), an optimisation function expression for the subproblem (code not given). The subproblem is solved (line 17), and if the maximal solution exceeds the capacity of the edge (line 18), a constraint is created.

As a result of solving the subproblem for each edge, a set of constraints are collected. Each constraint (line 19) is then added to the master problem (line 21).

If no constraints were collected (line 22), this means that no edge had its capacity exceeded, even in the worst case. Consequently the iteration terminates. If any constraints were added to the master problem, however, the iteration is repeated, with the newly tightened master problem.

The ECL^{i}PS^e language and implementation makes this decomposition very easy to state and to solve. ECL^{i}PS^e also supports many other solvers and hybridisation techniques. A description of the ECL^{i}PS^e Benders Decomposition library, for example, is available in [9].

Table 1. The Experimental Evaluation Result

Network (Nodes,Edges)	Flows	Obj	MIP cpu (Vars, Cstrs)	D(naive) cpu (MP)	D(0) cpu (MP)	D(ϵ) cpu (MP)
a (38, 172)	40	116	12.45 (25k,20k)	1.83 (1)	1.38 (1)	2.12 (3)
	54	132	18.77 (33k,26k)	4.05 (8)	3.38 (5)	3.52 (5)
b (38, 178)	54	132	18.90 (35k,27k)	4.27 (8)	3.39 (4)	3.85 (5)
c (38, 256)	140	328	108.97 (122k,91k)	16.07 (8)	13.63 (5)	14.15 (5)
d (50, 464)	126	226	TO (198k,146k)	710.47 (129)	35.77 (14)	29.08 (18)
	152	318	TO (236k,173k)	10060.64 (757)	48.16 (12)	72.56 (17)
	178	410	TO (274k,200k)	TO	517.60 (29)	783.94 (40)
e (50, 464)	286	546	TO (435k,317k)	TO	459.61 (50)	99.98 (17)
	350	714	TO (528k,383k)	TO	339.16 (21)	314.58 (19)
	418	890	TO (626k,453k)	TO	4033.98 (64)	2404.82 (43)
f (208, 676)	28	Fail	TO (73k,60k)	2673.38 (130)	79.66 (16)	76.63 (15)
	28	256	TO (73k,60k)	13350.71 (186)	252.44 (23)	706.21 (44)
g (208, 698)	28	76	257.64 (75k,62k)	20.83 (1)	20.90 (1)	20.76 (1)
h (208, 952)	44	108	361.84 (156k,123k)	50.79 (1)	50.69 (1)	52.87 (3)
i (208, 1186)	774	Fail	OOM (2910k,2153k)	2635.59 (22)	702.59 (3)	748.17 (3)
j (212, 734)	88	Fail	27.55 (223k,177k)	28.59 (1)	28.28 (1)	28.39 (1)
	104	Fail	33.06 (261k,207k)	TO	58.49 (4)	56.73 (3)
	106	338	TO (266k,210k)	TO	4602.80 (31)	7317.42 (40)
	154	Fail	TO (380k,300k)	TO	95.33 (5)	82.70 (2)
k (365, 1526)	142	Fail	71.22 (729k,565k)	185.76 (6)	139.47 (1)	140.31 (1)
	154	Fail	OOM (784k,606k)	TO	293.98 (4)	292.20 (3)
	178	422	OOM (900k,694k)	46930.15 (1572)	310.90 (12)	314.11 (12)

$\epsilon = 1.0e^{-5}$
cpu = seconds on Intel(R) Pentium(R) 4 CPU 2.00GHz
MP = number of iterations
OOM = Out of memory on Intel(R) Pentium(R) 4 with 1 Gig of main memory
TO = Timed out after 72000 seconds
Fail = Problem Unsatisfiable

7 Results and Discussion

ECL^{i}PS^e implementations of the models described in Sections 3–5 were run on a number of test instances on road networks of differing sizes, the smallest involving 38 junctions and 172 road segments and the largest 365 junctions and 1526 road segments. Our data is industrial and cannot be published in detail. For each road network the choice of blocked junctions with different in- and out-degrees results in problem instances having different numbers of flows to divert.

For each test instance the first 3 columns of Table 1 show each example road network with the number of nodes (junctions) and edges (road segments), the number of flows to reroute and optimal objective value (or Fail if no feasible solution exists). The remaining 4 columns show solution time for the four models with additionally the total number of variables and number of constraints in

the original MIP model and the number of master problem iterations for the decomposed models.

While there is some variation in the difficulty of individual instances for the different methods, the decomposed models outperform the MIP by at least an order of magnitude on average. It is striking how poorly MIP scales for this problem: while the MIP is able to solve only 9 of the instances within reasonable limits on execution time and memory usage, even the initial naive decomposition solves all but 8 instances within these limits, and the improved decomposed models solve all instances within a relatively small number of master problem iterations and relatively short time periods.

MIP focuses on integer inconsistencies which do not necessarily correlate closely with the real causes of inconsistency (*i.e.* overflow on an edge). CP enables us to solve a problem relaxation with whatever scalable solver is available (*e.g.* LP, or in the current master problem, MIP), and then to select the inconsistencies/bottlenecks at the problem level rather than at the encoded linear/integer level. This yields a much more problem-focused search heuristic.

This is reflected in the results obtained: all instances solvable by the MIP approach required only a few master problem iterations in the decomposed approach. In particular the 3 infeasible instances in data sets j and k for which MIP outperforms the decomposed approach are very over-constrained. Consequently they require little search in the MIP and few iterations in the decomposed models to prove infeasibility. Similarly the 6 feasible instances in data sets a, b, c, g and h are relatively loosely constrained and easily soluble by all methods. Even here however the combination of smaller component problem size and more problem-focused search in the decomposed approach yield order of magnitude improvements. Conversely the remaining instances which are neither very loosely constrained nor very over-constrained are very difficult for both the MIP approach and the naive decomposition due to the looseness of the cuts provided, but much more amenable to solution by the problem-focused tight cuts of the improved decomposition approaches.

Although this may suggest that MIP may be preferable for problems displaying certain characteristics, it is precisely those problems which are feasible but tightly constrained or infeasible but only slightly over-constrained that are most interesting in practice.

8 Conclusion

The diversion problem is very awkward in that the constraints involve a subsidiary optimisation function. The problem can be expressed as a single MIP, using the KKT condition to transform the embedded optimisation function into a set of constraints. Nevertheless the resulting MIP problem is very large and scales poorly.

A much better approach is presented by use of the decomposition strategy. In particular, tight cuts are created to improve the efficiency and scalability. The

experimental evaluation shows that decomposition can solve much larger scale problem instances.

References

1. Nemhauser, G. L., Wolsey, L. A.: Integer and Combinatorial Optimization, John Wiley & Sons, New York. (1988)
2. Jain, V., Grossmann, I.E.: Algorithms for Hybrid MILP/CP Models for a Class of Optimisation Problems. INFORMS Journal on Computing. **13** (2001) 258–276
3. Thorsteinsson, E.S.: Branch-and-Check: A Hybrid Framework Integrating Mixed Integer Programming and Constraint Logic Programming. In T. Walsh, editor, Principles and Practice of Constraint Programming – CP 2001, Springer. (2001) 16–30
4. Benders, J.F.: Partitioning Procedures for Solving Mixed-Variables Programming Problems. Numerische Mathematik. **4** (1962) 238–252
5. Hooker, J.N., Ottosson, G.: Logic-Based Benders Decomposition. Mathematical Programming. **96** (2003) 33–60
6. Cheadle, A. M., Harvey, W., Sadler, A., Schimpf, J., Shen, K., Wallace, M.: ECL^{i}PS^e. Technical Report 03-1, IC-Parc, Imperial College London, (2003)
7. ILOG: ILOG CPLEX 6.5 User's Manual, `http://www.cplex.com` (1999)
8. Dash Optimization: Dash Optimization Xpress–MP 14.21 User's Manual, `http://www.dashoptimization.com` (2003)
9. Eremin, A., Wallace, M.: Hybrid Benders Decomposition Algorithms in Constraint Logic Programming. In T. Walsh, editor, Principles and Practice of Constraint Programming – CP 2001, Springer. (2001) 1–15